Mobile Code Loading

Tom Mens

FWO Postdoctoral Fellow

Programming Technology Lab

Vrije Universiteit Brussel

Mobile code

- definition
    - any piece of software that may be transferred over a network to a different machine and executed
        - (it may also migrate during its execution)
- wide applicability of technology
    - electronic commerce
    - network management
    - software agents
    - distributed information retrieval
    - active networks
    - …

Problem

- Problem
  - execution of mobile code is slow
- Dominating slow-down factor
  - invocation latency
    - the time between application invocation and when execution of the program actually begins
    - due to: network delays, consistency checks, security checks, code decompression, compilation
  - network latency
    - time delay introduced by loading the code over the network
- Goal
  - speed up execution of mobile code
  - by reducing network latency

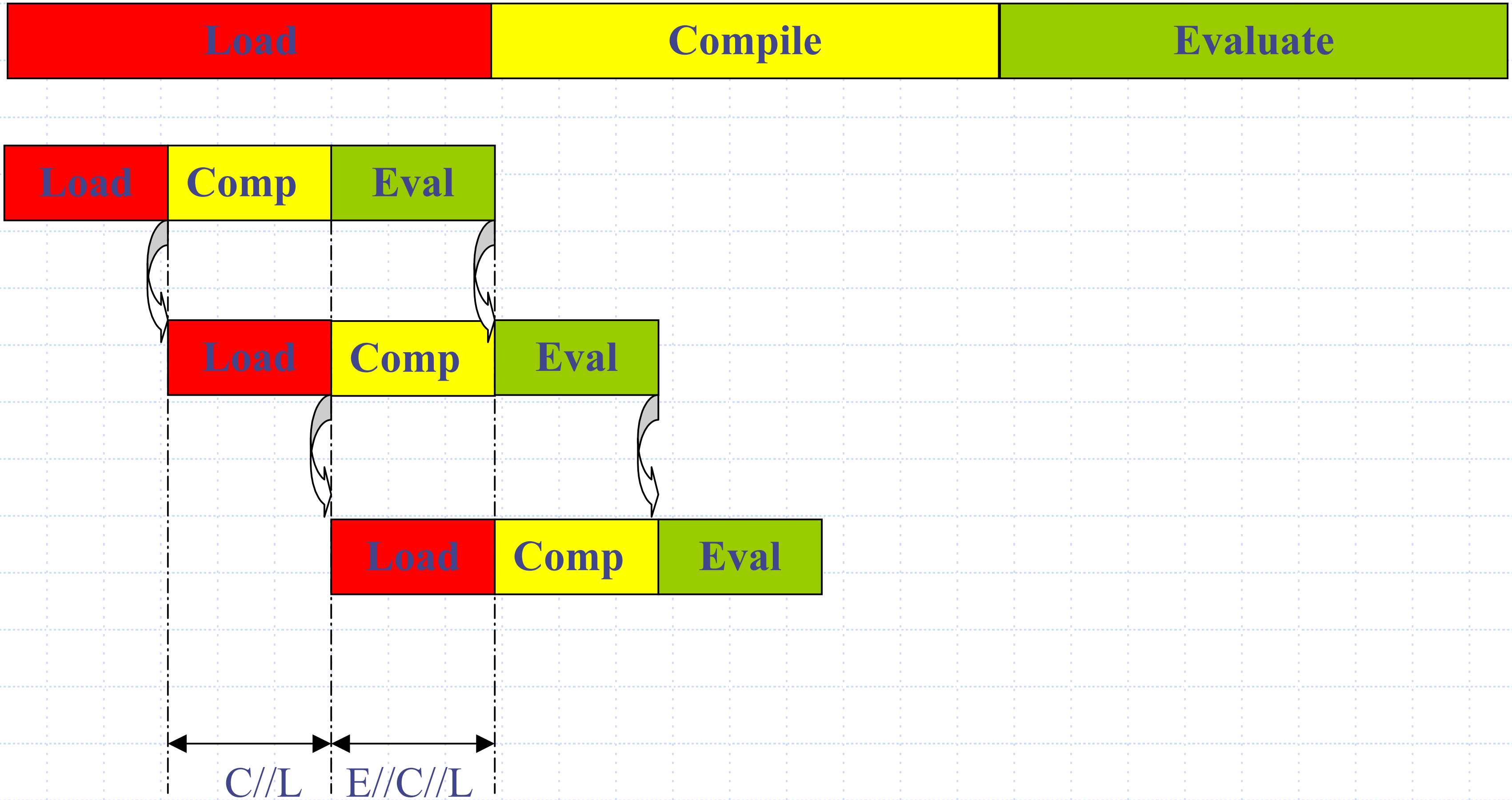
Insights

- network transmission time inherently slower than compilation and execution time for mobile applications
- gap between network speed and processor speed continues to widen
  - Law of Moore
- in mobile environments, performance is measured by invocation latency rather than overall execution time
  - user delays should be avoided

Proposed solutions

- transfer compressed code
  - compression/decompression is less time-consuming than transferring decompressed code
- reorder the loaded code
  - code that is needed first should be loaded first
    - requires code analysis
- exploit parallellism
  - loading, compilation and evaluation can be performed in parallel
    - different processors used for I/O and execution

Parallel Processing



Many factors involved

- programming language
  - Java (static typing), Smalltalk (dynamic typing)
- code representation
  - source code, parse tree, bytecode, machine code, compressed code
    - source code better for "simple" languages (e.g. Smalltalk)
    - bytecode better for "verbose" languages (e.g. Java)
- level of granularity
  - classes, methods

Many factors involved ctd.

- push versus pull technology
  - code on demand (e.g. Java dynamic class loading) vs. eager loading
- network bandwidth
  - e.g. LAN versus WAN, phone line versus cable modem, wireless communication
- compilation technique
  - e.g. just in time, ahead of time

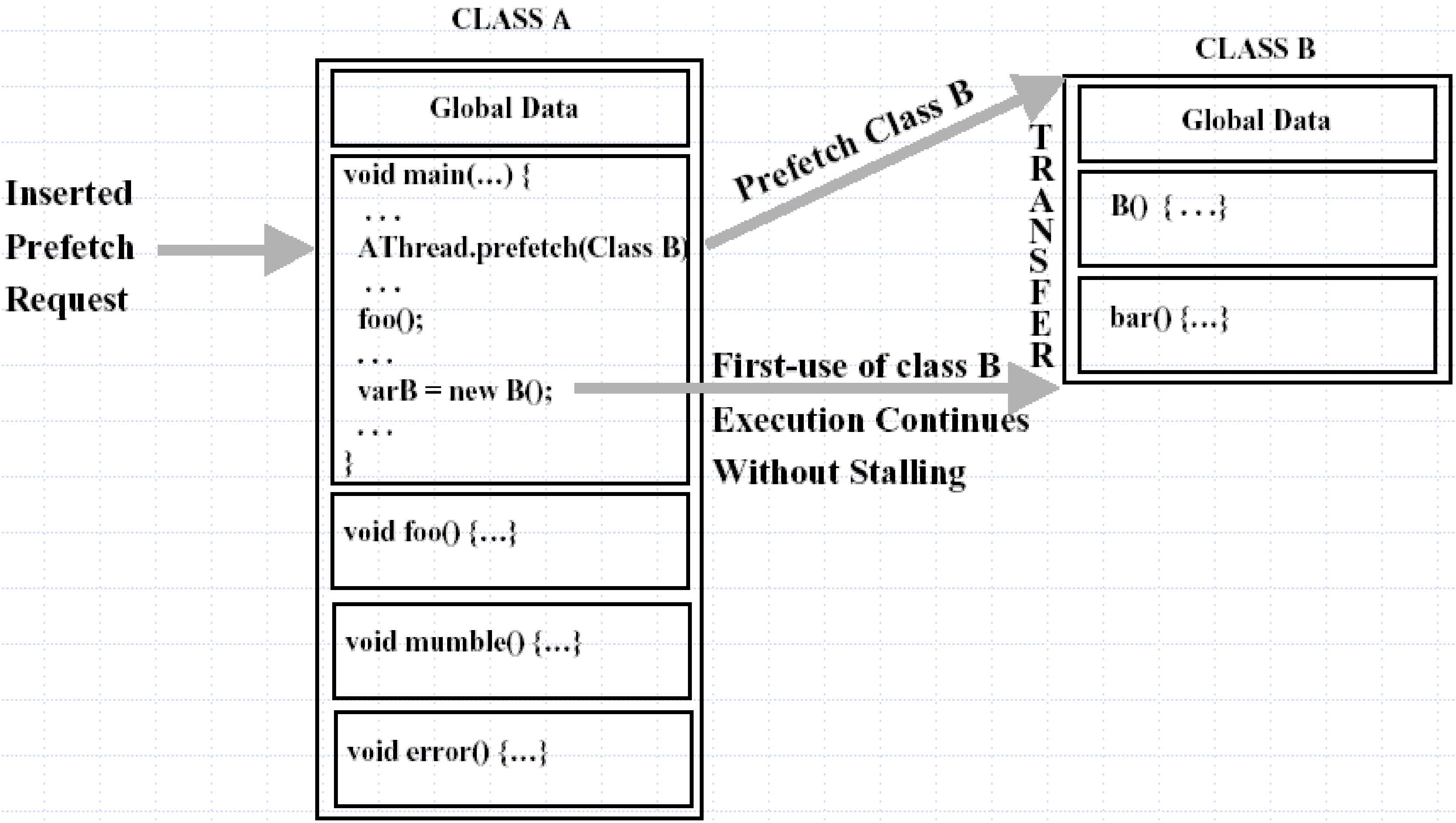
Different experiments

1. class file splitting and prefetching [Krintz&al1999]
   - Java bytecode, at class level
   - pull technology: code on demand using Java class loader
2. non-strict execution of mobile code [Krintz&al1998]
   - partial loading of Java class files, at method level
   - only simulation due to VM
3. interlaced code loading [Stoops&al2002]
   - Smalltalk source code, at method level
   - push technology: loading process triggers execution

1. Class splitting and prefetching - Technique

- class file splitting
  - partitions class file into hot and cold class file
  - avoid transfer of cold code that is rarely used
- class file prefetching
  - insert prefetch commands to overlap transfer with execution
    - optimise prefetch commands to maximise overlap
- trusted transfer
  - skip verification phase

1. Class prefetching - Technique

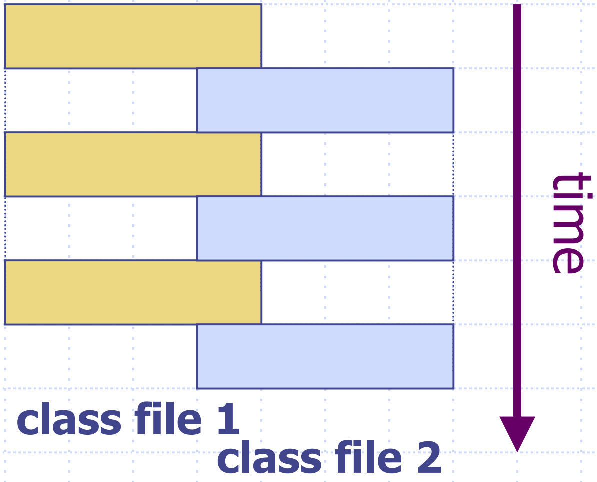


1. Class splitting and prefetching - Experiments

- code = bytecode
- language = Java
- granularity = class files
    - entire class must be loaded before its methods can be executed
- bandwidth = 2 simulations
  - 28.8 kbps (modem) and 1 Mbps (T1 link)
- case study = 7 applications
  - BIT, Jack, JavaC, JavaCup, Jess, Jlex, MPegAudio
- simulation results
  - splitting reduces startup time by 10%
  - splitting and prefetching reduces overall transfer delay by 25% to 30%
    - largest gains for T1 link

2. Non-strict execution for Java - Technique

- Two transfer techniques
  - parallel file transfer
    - loading multiple class files in parallel sharing bandwidth
  - interleaved file transfer
    - interleave loading of different class files



2. Non-strict execution for Java - Technique

- Reordering of methods and data
  - Transfer global data first
  - start verification process
  - predict first use ordering of methods in class
    - using static estimation based on control flow
    - using profiling based on training input sets
  - reorder methods
    - first local data, then code

2. Non-strict execution for Java - Experiments

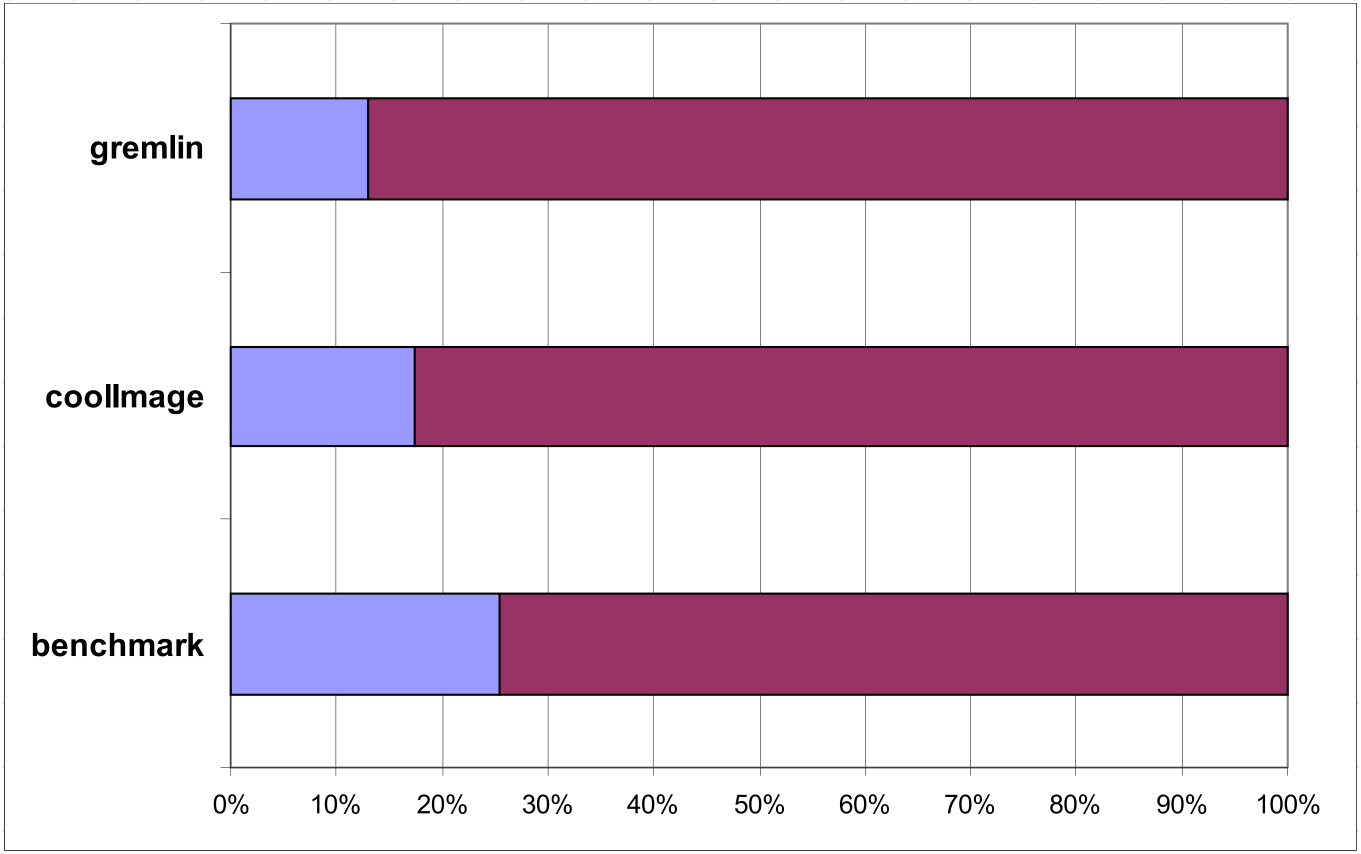
- code = bytecode
- language = Java
- granularity = method
- bandwidth = 2 simulations
  - 28.8 kbps (modem) and 1 Mbps (T1 link)
- case study = 6 applications
  - BIT, Hanoi, JavaCup, Jess, JHLZip, TestDes
- simulation results
  - simulation only because JVM security model requires complete class loading
  - average invocation latency reduction: 31 to 56%
  - average execution time reduction: 25 to 40%

3. Interlaced code loading - Technique

- Use JIT compilation of Smalltalk source code
- Reorder source code
  - Put GUI building code first to reduce user latency
  - Defer loading of low priority code
- Place semaphores in code to trigger execution during loading
  - put first semaphore after GUI building
  - put 3 semaphores evenly in rest of code

3. Interlaced code loading - Experiments

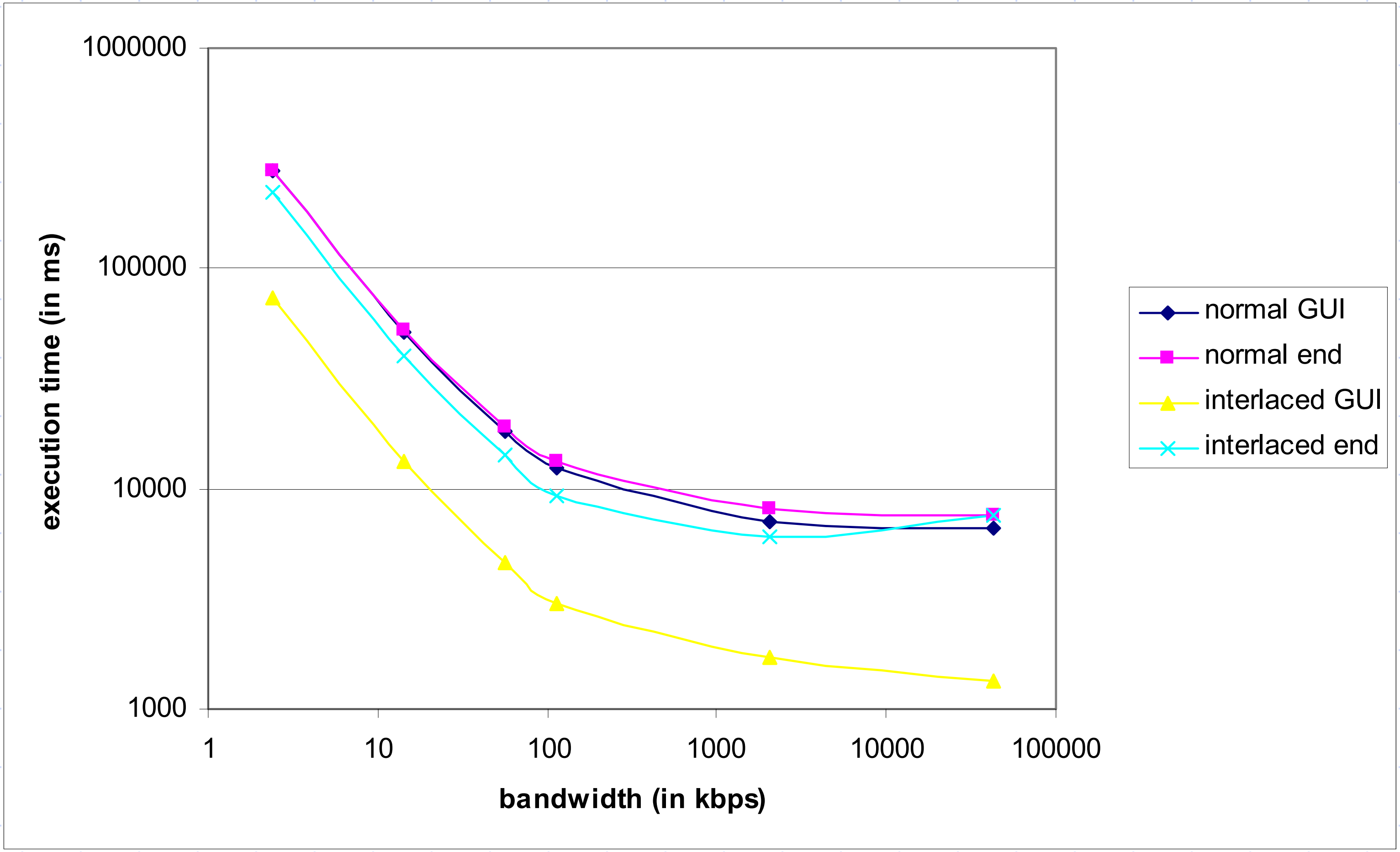
Percentage of code visited before GUI becomes available



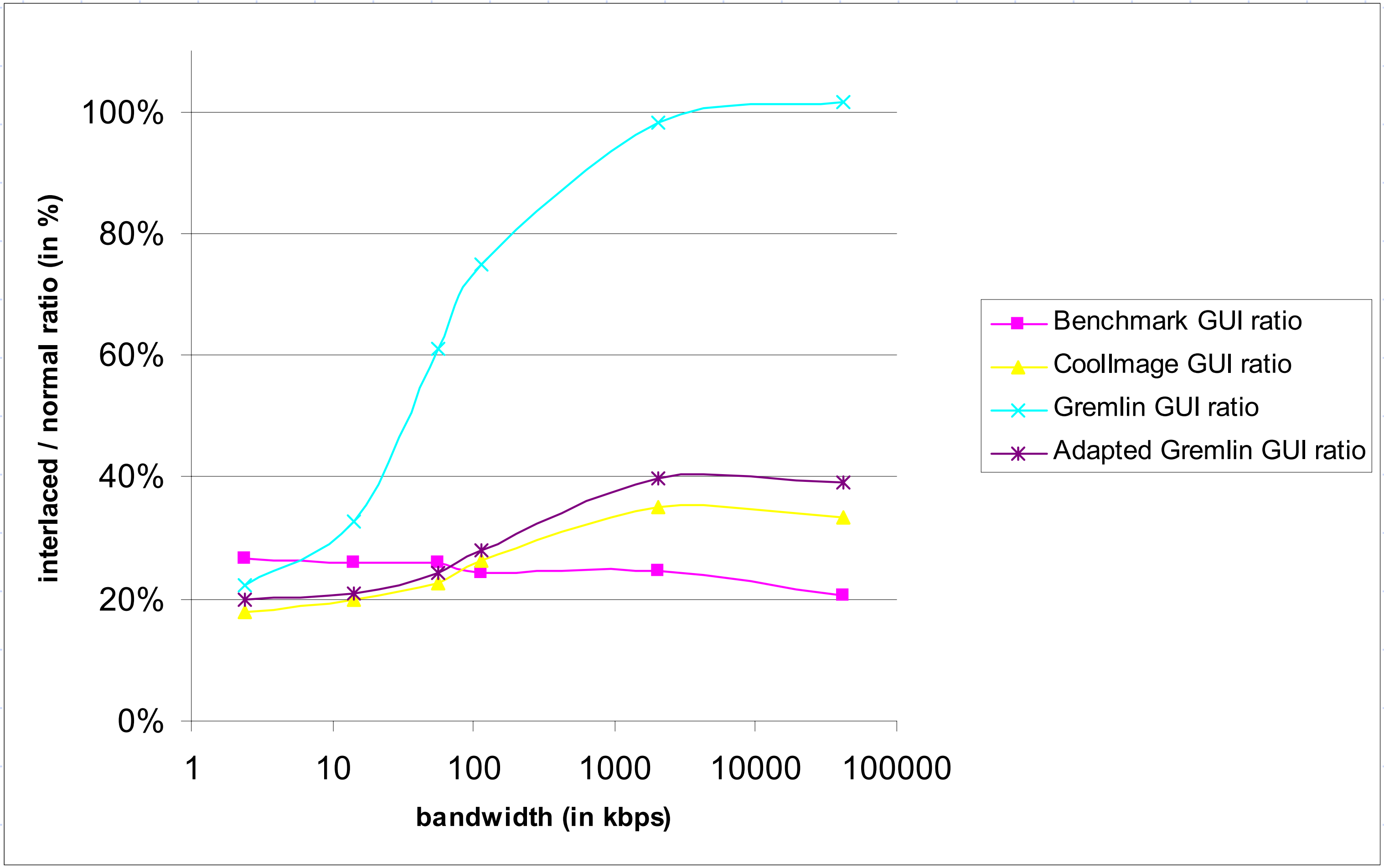
3. Interlaced code loading - Experiments

- code = source code
- language = Smalltalk (Visualworks)
- granularity = method
- bandwidth = 5 simulations
  - 2400 bps, 14.4 kbps (slow modem), 56 kbps (fast modem), 114 kbps (GPRS), 2 Mbps (UMTS)
- case study = 3 applications
  - Benchmark, CoolImage, Gremlin
- results =
  - reduction of user interface latency to 21 %
  - reduction of overall program execution time to 79 %

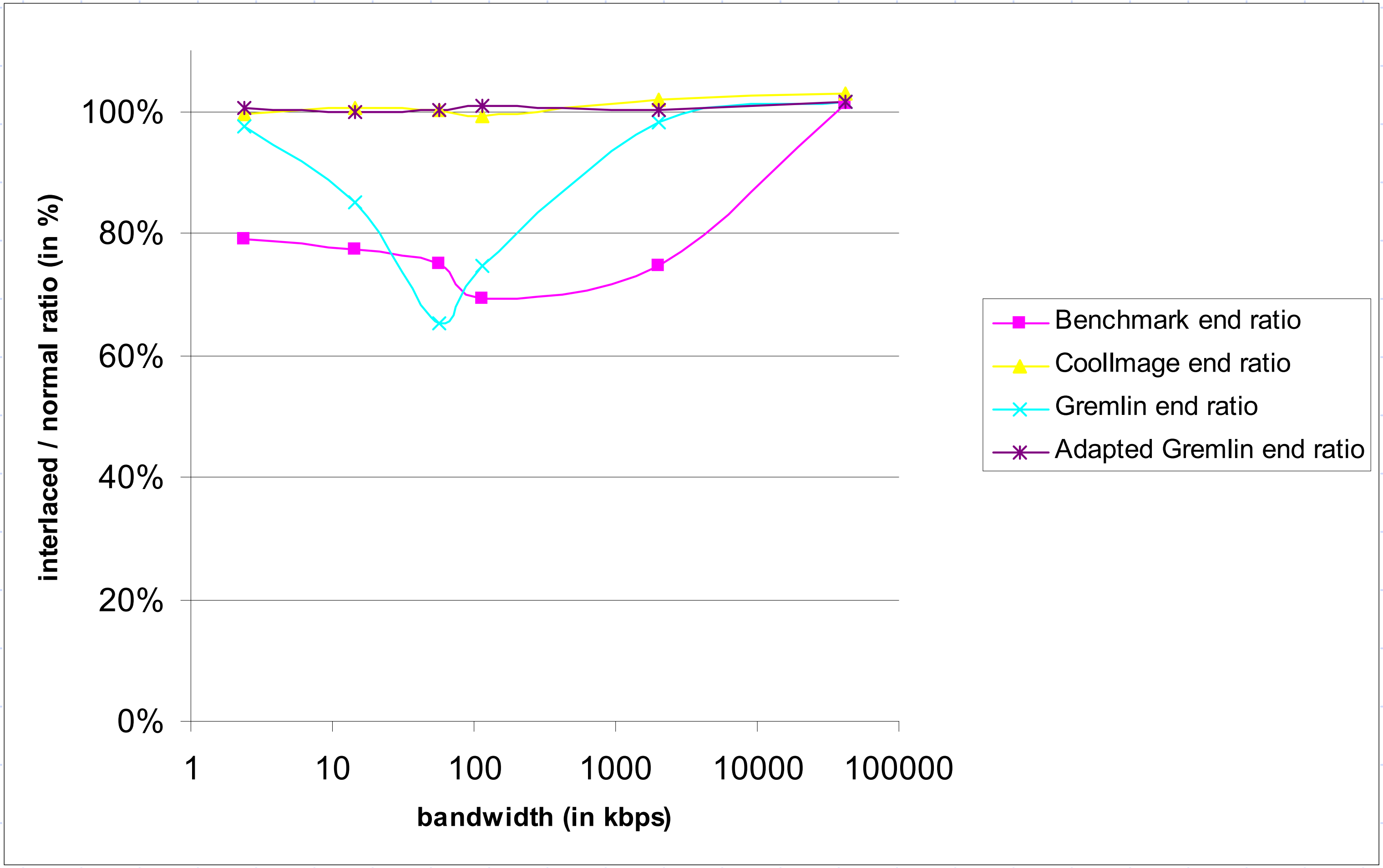
3. Interlaced code loading - Benchmark timing results



3. Interlaced code loading - Improved GUI building time



3. Interlaced code loading - Improved overall execution



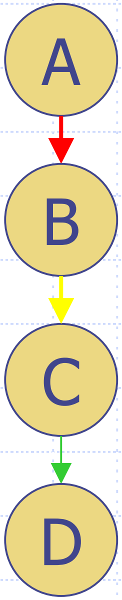
Conclusion

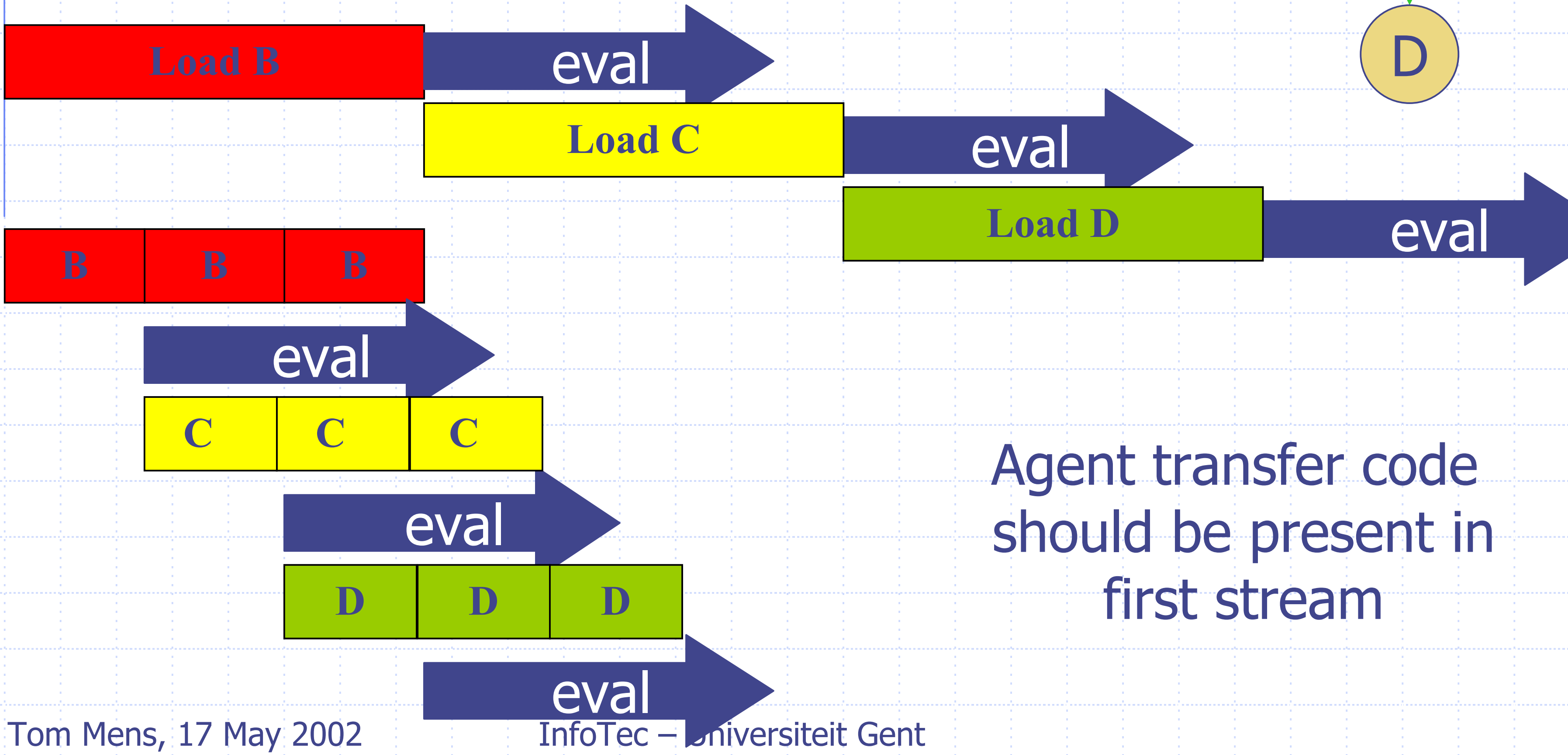
- Mobile code loading can be improved by
  - interleaving and parallellising loading/compilation/execution
  - reordering code and data
  - loading different code parts in parallel over same channel
  - (compressing code and data)
- Benefits
  - generally applicable
  - reduces invocation latency
  - reduces user interface latency
  - speeds up program execution

  - Many variants of technique possible depending on a variety of factors

Mobile agent hopping

- Mobile agent that executes some code in different nodes of a network





Agent transfer code should be present in first stream

References

- About reducing network latency
  - C. Krintz, B. Calder, H.B. Lee, B.G. Zorn. Overlapping execution with transfer using non-strict execution for mobile programs. Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems, October, 1998
  - C. Krintz, B. Calder, U. Hölzle. Reducing transfer delay using class file splitting and prefetching. Proc. Int. Conf. OOPSLA, November, 1999
  - L. Stoops, T. Mens. Interlaced code loading for mobile systems. Mobility WS, ECOOP 2002